send_lowat

Context: `http`, `server`

This is an option that allows you to make use of the `SO_SNDLOWAT` flag for TCP sockets under FreeBSD only. This value defines the minimum number of bytes in the buffer for output operations.

Syntax: Numeric value (size)

Default value: `0`

reset_timedout_connection

Context: `http`, `server`, `location`

When a client connection times out, its associated information may remain in memory depending on its state. Enabling this directive will erase all memory associated with the connection after it times out.

Syntax: `on` or `off`

Default value: `off`

Paths and documents

This section describes the directives that configure the documents that should be served for each website, such as the document root, the site index, error pages, and so on.

root

Context: `http`, `server`, `location`, `if`. Variables are accepted.

This directive defines the document root containing the files that you wish to serve to your visitors.

Syntax: Directory path

Default value: `html`

```
root /home/website.com/public_html;
```

alias

Context: `location`. Variables are accepted.

`alias` is a directive that you place in a `location` block only. It assigns a different path for Nginx to retrieve documents for a specific request. As an example, consider the following configuration:

```
http {
   server {
       server_name localhost;
       root /var/www/website.com/html;
       location /admin/ {
        alias /var/www/locked/;
      }
   }
}
```

When a request for `http://localhost/` is received, files are served from the `/var/www/website.com/html/` folder. However, if Nginx receives a request for `http://localhost/admin/`, the path used to retrieve the files is `var/www/locked/`. Moreover, the value of the document root directive (`root`) is not altered. This procedure is invisible in the eyes of dynamic scripts.

Syntax: Directory (do not forget the trailing `/`) or file path

error_page

Context: `http`, `server`, `location`, `if`. Variables are accepted.

This allows you to affect URIs to the HTTP response code and optionally, to substitute the code with another.

Syntax: `error_page code1 [code2…] [=replacement code] [=@block | URI]`

Examples :

```
error_page 404 /not_found.html;
error_page 500 501 502 503 504 /server_error.html;
error_page 403 http://website.com/;
error_page 404 @notfound; # jump to a named location block
error_page 404 =200 /index.html; # in case of 404 error, redirect to
index.html with a 200 OK response code
```

if_modified_since

Context: `http`, `server`, `location`

This defines the way Nginx handles the `If-Modified-Since` HTTP header. This header is mostly used by search engine spiders (such as Google web crawling bots). The robot indicates the date and time of the last pass. If the requested file has not been modified since that time, the server simply returns a `304 Not Modified` response code with no body.

This directive accepts the following three values:

- `off`: Ignores the `If-Modified-Since` header.
- `exact`: Returns `304 Not Modified` if the date and time specified in the HTTP header are an exact match with the actual requested file modification date. If the file modification date is anterior or ulterior, the file is served normally (`200 OK` response).
- `before`: Returns `304 Not Modified` if the date and time specified in the HTTP header is anterior or equal to the requested file modification date.

Syntax: `if_modified_since off | exact | before`

Default value: `exact`

index

Context: `http`, `server`, `location`. Variables are accepted.

This defines the default page that Nginx will serve if no filename is specified in the request (in other words, the index page). You may specify multiple filenames, and the first file to be found will be served. If none of the specified files are found, and if the `autoindex` directive is enabled (check the `HTTP Autoindex` module), Nginx will attempt to generate an automatic index of the files. Otherwise, it will return a `403 Forbidden` error page. Optionally, you may insert an absolute filename (such as `/page.html`, based from the document root directory) but only as the last argument of the directive.

Syntax: `index file1 [file2…] [absolute_file];`

Default value: `index.html`

```
index index.php index.html index.htm;
index index.php index2.php /catchall.php;
```

recursive_error_pages

Context: `http`, `server`, `location`

Sometimes, an error page itself served by the `error_page` directive may trigger an error; in this case, the `error_page` directive is used again (recursively). This directive enables or disables recursive error pages.

Syntax: `on` or `off`

Default value: `off`

try_files

Context: `server`, `location`. Variables are accepted.

This attempts to serve the specified files (arguments 1 to N-1). If none of these files exist, it jumps to the respective named `location` block (last argument) or serves the specified URI.

Syntax: Multiple file paths followed by a named `location` block or a URI

Example:

```
location / {
    try_files $uri $uri.html $uri.php $uri.xml @proxy;
}
# the following is a "named location block"
location @proxy {
    proxy_pass 127.0.0.1:8080;
}
```

In this example, Nginx tries to serve files normally. If the requested URI does not correspond to any existing file, Nginx appends `.html` to the URI and tries to serve the file again. If it fails again, it tries with `.php`, and then with `.xml`. Eventually, if all of these possibilities fail, another `location` block (`@proxy`) handles the request.

You may also specify `$uri/` in the list of values in order to test for the existence of a directory with that name.

Client requests

This section documents the way that Nginx handles client requests. Among other things, you are allowed to configure the keep-alive mechanism behavior and, possibly, logging the client requests into files.